Attorney Docket No.: 7451.0034-00

APPLICATION FOR UNITED STATES PATENT

TRUST MANAGEMENT SYSTEMS AND METHODS

By Inventors:

Stephen T. Weeks
1261 Lakeside Drive #3197
Sunnyvale, California 94086
A Citizen of the United States

Xavier Serret-Avila
900 Pepper Tree Lane #9211
Santa Clara, California 95051
A Citizen of France

Assignee: InterTrust Technologies Corporation
4750 Patrick Henry Drive
Santa Clara, CA 95054

Status: Large Entity

TRUST MANAGEMENT SYSTEMS AND METHODS

RELATED APPLICATIONS

[001] This application claims the benefit of U.S. Provisional Application No. 60/205,402, entitled "Trust Management Systems and Methods," filed May 19, 2000, which is hereby incorporated by reference in its entirety.

COPYRIGHT AUTHORIZATION

[002] A portion of the disclosure of this patent document contains material which is subject to copyright protection. The copyright owner has no objection to the facsimile reproduction by anyone of the patent document or the patent disclosure, as it appears in the Patent and Trademark Office patent file or records, but otherwise reserves all copyright rights whatsoever.

FIELD OF THE INVENTION

[003] The present invention relates generally to the management of digital content and electronic transactions. More specifically, systems and methods are disclosed for making trust management decisions.

BACKGROUND OF THE INVENTION

[004] In modern computing systems, it is often desirable to limit access to electronic content or processing resources, or to allow only certain parties to perform certain actions. A number of techniques have been used or proposed for enabling such control. One set of techniques makes use of digital certificates. A digital certificate can be viewed as an authorization to perform an action or to access a resource. An entity may possess numerous certificates. In order to determine whether an entity is authorized to perform an action, one looks at the certificates in the entity's possession and determines whether authorization has been granted. Because authorization may be implicit or delegated (e.g., one entity may authorize another entity to issue authorizations on its

behalf), it is often necessary to evaluate multiple certificates to determine whether a request for system resources should be granted.

[005] Trust management systems define languages for expressing authorizations and access control policies, and provide trust management engines for determining when a particular request is authorized. The task of the trust management engine will typically involve evaluating both the request and a set of certificates associated with the request and/or the requestor. In some conventional trust management systems, the trust management engine makes decisions using a process similar to that shown in Fig. 1. As shown in Fig. 1, trust management engine 102 accepts a request 104 to perform an action or to access a resource. The trust management engine also accepts a set of certificates 106 associated with the requestor or the request, and some indication 108 of the identity of the owner or "root" associated with the requested resource (i.e., the authorizing principal from whom permission to access the resource must ultimately flow). Next, the trust management engine performs a certificate path discovery process 110 on the group of certificates 106 to determine the way in which the certificates should be arranged in order to facilitate the remainder of the trust management decision process. The result of certificate path discovery is an ordered list of certificates 112. The ordered list of certificates 112 is then reduced by successively combining adjacent certificates in the list 114. The end result of the reduction process is a certificate 116 representing the combination of each of the assertions expressed by the certificates in the original group. This certificate is then evaluated 118 to determine whether the original request should be granted or denied 120.

[006] A problem with approaches such as the one shown in Fig. 1 is that the computation involved can be relatively costly.

SUMMARY OF THE INVENTION

[007] Systems and methods are provided for reducing the cost and/or complexity of the trust management decision process. The systems and methods of the present invention are applicable to the Simple Public Key Infrastructure (SPKI) architecture and to a wide variety of other trust management architectures. It should be appreciated that

the present invention can be implemented in numerous ways, including as a process, an apparatus, a system, a device, a method, a computer readable medium, or as a combination thereof. Several inventive embodiments of the present invention are described below.

[008] In one embodiment, a method is described for controlling access to computing resources. A request for a computing resource is obtained, as is a group of certificates. The certificates express authorizations for performing certain actions or for accessing certain resources, and are issued by system principals. The principals are identified and assigned an initial state. Next, the certificates are iteratively evaluated, and the states of the principals updated, until a fixpoint is reached or until a predefined one of the principals is determined to have authorized the request.

[009] In another embodiment, a system is provided for controlling access to electronic content and processing resources. The system includes means for receiving a request from a principal to access a piece of electronic content or a processing resource, and means for collecting a set of certificates relating to the request, the requesting principal, or the electronic content or processing resource. The system further includes means for identifying a root principal from whom authorization is needed in order to grant the request. A least fixpoint computation is used to determine whether the root principal has authorized the requesting principal to access the piece of electronic content or the processing resource.

[010] In yet another embodiment, a system is provided for controlling access to electronic resources. A first computer system is operable to process requests for system resources and to grant or deny access. The first computer system includes a network interface for receiving digital certificates from other systems and for receiving requests to access electronic resources. The first computer system also includes a memory unit for storing electronic resources and one or more certificates relating thereto. The first computer system includes a trust management engine for processing digital certificates and requests for electronic resources, and for making access control decisions based on least fixpoint computations. In some embodiments, requests for system resources may be received from a second computer system. A third computer system might generate a

digital certificate that authorizes the second computer system to access the system resource, and might send this digital certificate to the second and/or first computer systems. In addition, a fourth computer system might generate another digital certificate that authorizes the third computer system to make the authorization described above.

[011] In another embodiment, a computer program product is provided for making trust management determinations. The computer program includes computer code for obtaining requests to perform certain actions, and computer code for obtaining a group of authorizations associated with the requested actions. The computer program also includes computer code for identifying a set of principals associated with the authorizations, and for initializing the state of each principal. Computer code is also provided for evaluating the authorizations based on the state of the principals, and for updating the state of the principals. The computer code for evaluating the authorizations and for updating the state of the principals is repeated until a fixpoint is reached or until a predefined principal is deemed to authorize the request. The computer program is stored on a computer-readable medium such as a CD-ROM, DVD, MINIDISC, floppy disk, magnetic tape, flash memory, ROM, RAM, system memory, web server, hard drive, optical storage, or a data signal embodied in a carrier wave.

[012] In another embodiment, a method for performing trust management computations is described. A group of certificates is collected, each of the certificates including at least one authorization. The authorizations contained in the certificates are expressed using a structure that satisfies certain predefined properties. In one embodiment, this structure is a lattice. The certificates are expressed as functions, and a least fixpoint is computed using these functions. In one embodiment, the functions are monotone. Trust management decision are made using the results of the least fixpoint computation.

[013] In another embodiment, a lattice is constructed that represents the authorizations granted by each of the authorizing principals in a system. To facilitate the construction of this lattice, the licenses derived from the authorizations are represented by monotone functions. A fixpoint algorithm is used to make trust management decisions based on the lattice. In some embodiments, only a portion of the fixpoint is computed,

the portion being sufficient to determine if the particular request is authorized. Thus, trust management decisions can be made without the necessity of performing certificate path discovery.

[014] In another embodiment, a framework is presented that enables trust management systems to be concisely specified, which helps in comparing existing systems and in designing new systems. The semantics of a trust management engine are defined as a least fixpoint in a lattice, and this definition is used to implement a trust management engine. Trust management engines implemented in this manner are able to make trust management decisions more efficiently than trust management engines that rely on certificate path discovery and tuple reduction.

[015] These and other features and advantages of the present invention will be presented in more detail in the following detailed description and the accompanying figures which illustrate by way of example the principles of the invention.

BRIEF DESCRIPTION OF THE DRAWINGS

[016] The present invention will be readily understood by the following detailed description in conjunction with the accompanying drawings, wherein like reference numerals designate like structural elements, and in which:

[017] Fig. 1 illustrates a conventional technique for making trust management decisions.

[018] Fig. 2 illustrates a system for practicing embodiments of the present invention.

[019] Fig. 3 illustrates a method for making trust management decisions in accordance with an embodiment of the present invention.

[020] Fig. 4 illustrates a trust management engine in accordance with an embodiment of the present invention.

[021] Figs. 5A, 5B, 5C, 5D, 5E, 5F, 5G, 5H, and 5I illustrate the process of making trust management decisions in accordance with embodiments of the present invention.

[022] Fig. 6 illustrates a method for making trust management decisions in accordance with an embodiment of the present invention.

[023] Fig. 7 illustrates another method for making trust management decisions in accordance with an embodiment of the present invention.

[024] Fig. 8 illustrates another method for making trust management decisions in accordance with an embodiment of the present invention.

[025] Fig. 9 illustrates a method for making trust management decisions in an embodiment that makes use of dependency graphs.

[026] Fig. 10 illustrates a set of dependency graphs for use in making trust management decisions.

[027] Fig. 11 illustrates a system for practicing embodiments of the present invention.

[028] Fig. 12 shows a set of SPKI name assertions and a method for processing these assertions in accordance with an embodiment of the present invention.

[029] Fig. 13 shows a set of SPKI authorization assertions and a method for processing these assertions in accordance with an embodiment of the present invention.

[030] Fig. 14 illustrates a computer system that can be used to practice embodiments of the present invention.

[031] Fig. 15 shows examples of licenses in accordance with embodiments of the present invention.

[032] Fig. 16 shows examples of least fixpoint computations in accordance with embodiments of the present invention.

[033] Fig. 17 illustrates a framework for expressing trust management systems in accordance with an embodiment of the present invention.

[034] Fig. 18 shows an expression of SPKI using the framework of Fig. 17.

[035] Fig. 19 shows an expression of Keynote using the framework of Fig. 17.

DETAILED DESCRIPTION

[036] A detailed description of the invention is provided below. While the invention is described in conjunction with several embodiments, it should be understood

that the invention is not limited to any one embodiment. On the contrary, the scope of the invention is limited only by the appended claims and encompasses numerous alternatives, modifications, and equivalents. For example, while embodiments are described in the context of making trust management decisions using the Simple Public Key Infrastructure (SPKI) and Keynote, those of ordinary skill in the art will recognize that the disclosed systems and methods are readily adaptable for broader application. For example, without limitation, the present invention could be readily applied in the context of other trust management systems. In addition, while numerous specific details are set forth in the following description in order to provide a thorough understanding of the present invention, the present invention may be practiced according to the claims without some or all of these details. Finally, for the purpose of clarity, certain technical material that is known in the art has not been described in detail in order to avoid obscuring the present invention. For example, reference will be made to a number of terms and concepts that are well-known in the field of cryptography. Background information on cryptography can be found, for example, in Menezes et al., *Handbook of Applied Cryptography* (CRC Press 1996)("Menezes"); and Schneier, *Applied Cryptography*, 2d ed. (John Wiley & Sons 1995). Use will also be made, on occasion, of certain mathematical notation. Readers unfamiliar with this notation should refer to Appendix A.

[037] The present invention provides systems and methods for making efficient trust management decisions. The processing logic of the present invention is operable to accept requests for system resources, authorizations or certificates, and the identity of one or more root authorities. To determine whether a request should be granted, the processing logic identifies a set principals from whom authorization may flow, and interprets each of the authorizations or certificates as a function of the state of one or more of the principals. The processing logic iteratively evaluates the functions represented by the certificates, updates the state of the principals, and repeats this process until a reliable determination can be made as to whether the request should be granted or denied. In a preferred embodiment the certificates are evaluated until the state of the root

authority indicates that the request should be granted, or until further evaluation of the certificates is ineffective in changing the state of the principals.

[038] For purposes of illustration, the structure and operation of an embodiment of the present invention will be described in the context of a user ("Alice") attempting to access a piece of content on a remote system. As shown in Fig. 2, the user's system 202 may, for example, establish a connection with a remote system or server 204 via a network 206 such as the Internet. The user may then request permission to read or modify content 208 stored on the server.

[039] As shown in Fig. 3, the server, upon receiving a user's request to access content or other system resources (302), evaluates whether the user is authorized to access the requested content or resources. Typically, the server program responsible for controlling access to the server's resources will maintain records of the access control policies associated with various content items or other resources. These records specify the party or parties from whom authorization must be obtained, the nature of the authorizations, and/or the other conditions that must be satisfied in order for access to be granted. To determine whether the user is authorized to access the requested content or resources, the server identifies the entity from whom permission to access the requested item is needed (i.e., the root authority)(304), and collects the certificates that are needed to make the determination (306). As shown in Fig. 2, a trust management engine 210 is then used to examine the available authorizations/certificates to determine whether the root has granted the user permission to access the requested item. For example, if the requested item is a document, the author's permission may be needed before server 204 will allow a user 202 to modify the document. In a preferred embodiment, the trust management engine treats the certificates as functions and computes a least fixpoint (308). If the trust management engine determines that the root principal has authorized the user to perform the requested action (i.e., a "Yes" exit from block 310), then the server allows the user to access the requested resource or perform the requested action (312). Otherwise, the request is denied (314).

[040] Note that server 204 can obtain the certificates 212 needed to evaluate a user's request 211 in any of a variety of ways. For example, server 204 might obtain one

or more of these certificates 212a directly from the user, the user having obtained these certificates previously from various other principals in the system (e.g., principal 218). Alternatively, or in addition, the server may already have obtained some of the certificates before the user made his or her request, and/or the server may contact other relevant principals (e.g. 214, 216) and request such certificates after the user's request is received. In short, certificates can be obtained in any suitable manner.

[041] Note, too, that the certificates themselves can take any of a wide variety of forms. For example, conventional digital certificates typically contain an indication of the nature of the authorization that is being granted and the identity of the entity to whom the authorization is directed. For example, the certificate may contain the public key of the authorized entity. The authorized entity can then "redeem" the authorization by demonstrating its possession of the corresponding private key. Conventional certificates are typically signed with the private key of the principal granting the authorization, thus enabling a party who verifies the signature with the principal's public key to be confident that the certificate originated from the principal and has not been modified.

[042] The operation of an exemplary trust management engine will now be described in more detail. The discussion will occasionally refer to *principals, authorizations, authorization maps, licenses,* and *assertions.* In general terms, a principal can be viewed as an entity that may make or authorize requests. Principals should be distinguishable from one another, and in some embodiments may be identified by a public key. An authorization expresses the permissions granted by a principal. Authorizations can form a lattice. In an illustration presented in Figs. 5A-5I below, for example, the authorizations form the lattice {N, R, W, RW}, where N means that no authorization is given, R denotes authorization to read a file, W denotes authorization to write to a file, and RW denotes authorization to read and write. Here, the level of authorization increases from left to right – that is, N represents a lesser level of authorization than R, R and W have the same level of authorization, and RW represents the greatest level of authorization

[043] An authorization map is a function that maps principals to authorizations, and that describes the authorizations made by each principal in a trust management

decision. In the examples that will be described in connection with Figs. 5A – 5I below, for example, the authorization maps express the authorization each principal grants to another principal named "Alice". A license grants authorization expressed as a monotone function from authorization maps to authorizations. Intuitively, the meaning of license, *l*, is that if principals grant authorizations as given by an authorization map, *m*, then *l* grants the authorization *l(m)*. Here, monotonicity means that the authorizations granted by a license increase (or stay the same) as a consequence of an increase in other principals' authorizations. Monotonicity gives a sensible meaning to a collection of assertions so that the trust management engine is well-defined.

[044] An assertion is an expression of authorization made by a principal. Assertions can be thought of as an abstraction of digital certificates. Formally, an assertion can be viewed as consisting of a principal and a license. In practice, assertions will typically be signed by the public key of the issuing principal, and may require signature checking before being processed by the trust management engine.

[045] A trust management engine makes a decision based on a request, an authorizing principal, and a set of assertions. The authorizing principal reflects the policy governing the request. A request is expressed as an element of the authorization lattice. A trust management engine takes a set of assertions made by various principals, some of whom may delegate to each other, and finds a coherent authorization map that represents the authorizations of these principals. The trust management engine then determines if the request requires less authorization than is granted by the authorizing principal, *p*. For example, in the lattice described above (i.e., {N, R, W, RW}), if Alice would like to write to a file under Bob's control, and the assertions a_1, a_2, and a_3 are available, the trust management engine would compute the least fixpoint using the request and these assertions to determine whether Bob has authorized Alice to write. The assertions may reside in the trust management engine's local machine or network, or may be presented to the trust management engine's local machine by, e.g., the principal making the request.

[046] Note that the trust management engine does not necessarily need to compute the entire fixpoint for the given assertions. To prove that a request is authorized, the engine simply finds an authorization map in which the requested action is

authorized. Finding such an authorization map may be easier than computing the entire fixpoint, since (a) the trust management engine may be able to work in a lattice in which elements are more compactly represented and licenses more efficiently computed, and (b) the engine may terminate the fixpoint computation once a large enough authorization map has been found. This is illustrated below in the discussion of SPKI.

[047] The expressiveness of a trust management system will typically depend on the choice of a lattice of authorizations, and on the choice of a subset of the monotone functions to be used as licenses. In general, the optimal choices will vary from application to application; however, it should be understood that for purposes of practicing the present invention any suitable choices can be made. Similarly, the choice of a representation for authorizations, and a language for licenses, such that the desired monotone functions can be expressed concisely, and such that the fixpoint can be computed efficiently, will also vary from application to application. However, any suitable choice or choices can be made without departing from the principles of embodiments of the present invention.

[048] Fig. 4 shows a trust management engine. Trust management engine 400 is responsible for taking a set of certificates/assertions 402 and a specification of a root principal 404, and determining whether these certificates indicate that the root principal authorizes a requested action 406. As indicated above, trust management decisions can be made by interpreting authorizations as monotone increasing functions of the authorization state of the principals from whom authorization is to be obtained, and evaluating a least fixpoint of the principals' assertions (or an approximation of the least fixpoint).

[049] Several examples of this process are described below in connection with Figs. 5A-5I. To facilitate explanation, in these examples each principal in the system will be associated with a single authorization that describes the permissions granted by that principal. In addition, the authorization lattice will specify the authorizations granted to a single principal – Alice – for a particular file. It should be appreciated, however, that these constraints are not necessary, and that the concepts and techniques that are illustrated by these examples can be readily applied to systems in which principals are

associated with multiple authorizations and/or the authorization lattice specifies authorizations for multiple principals and/or for multiple resources.

[050] Figs. 5A-5I show some example computations of a least fixpoint for a set of assertions 530a-534i. The order of the assertions is not important. In the examples shown in Figs. 5A-5I, there are four principals – Alice, Bob, Carl, and Dave – and it is desired to determine what authorizations Bob, Carl, and Dave have granted to Alice. The authorizations or permissions that can be granted in these examples are the ability to read from and/or write to a file (i.e., {N, R, W, RW}). In Figs. 5A-5I, the fixpoint computation 502a – 502i is shown as a sequence of authorization maps, one per row, where each column contains the authorizations made by a principal. Each row contains an authorization map consisting of the least upper bound of all of the assertions applied to the authorization map in the previous row. The final row is the least fixpoint authorization map of the set of assertions. The examples show how least fixpoint computation can express concepts like path validation, chaining and five-tuple reduction, and inter-assertion communication.

[051] In the example shown in Fig. 5A, there is only one relevant certificate 530a. It indicates that Bob has authorized Alice to write. Here, the fixpoint computation is simple. The state of the principals is initialized to "No" (i.e., row 1 of table 502a). Next, the certificate is evaluated and the authorization map is updated to indicate that Bob has authorized writing (row 2). Since further evaluation of the certificate will not change the state of any of the principals, a fixpoint has been reached, and a trust management decision can be made. For example, if Alice were seeking Bob's permission to write to the file, her request would be granted. However, if Alice were seeking to read from the file, or if she wanted to write to the file but needed the permission of Carl or Dave, then her request would be denied.

[052] Fig. 5B shows an example in which Bob has issued two certificates. One certificate indicates that Bob has given Alice permission to read, the other certificate indicates that Bob has give her permission to write. The least fixpoint is computed in the manner described above.

[053] In Fig. 5C, Alice has two certificates: one indicates that Bob has granted her permission to write 530c; the other indicates that Carl has granted her permission to do whatever Bob has authorized her to do 532c. As shown in Fig. 5C, to compute the least fixpoint, the state of each of the principals is initialized to "No" (i.e., row 1 of table 502c). Next, each of the certificates is evaluated based on the current state of the principals (i.e. the state specified in row 1 of table 502c), and any updated state information is placed in the next row of the table (i.e., row 2). As shown in Fig. 5C, evaluation of the certificates results in table 502c being updated to show that Bob has authorized write access (see row 2). Note, however, that row 2 does not indicate that Carl has authorized write access. This is because certificate 532c indicates that Carl will authorize whatever Bob authorizes, but row 1 indicates that Bob has not authorized anything. However, when the certificates are evaluated again – this time using the data in row 2 of the table – the authorization map will be updated to show that Carl has granted write access (row 3). Since further evaluation of the certificates using the data contained in row 3 of the table will not lead to further changes in the authorization map, the least fixpoint has been reached and additional processing is unnecessary.

[054] Fig. 5D shows a situation in which the least fixpoint computation 502d indicates that none of the parties has granted authorization. Here, Bob has indicated that he will authorize whatever Carl has authorized 530d; however, Carl has indicated that he will only authorize what Bob has authorized 532d. Because the state is initialized to "No", evaluation of the certificates will not change the state, and thus the least fixpoint consists of the first row of table 502d.

[055] Figs. 5E and 5F illustrate the concept of constrained delegation. In Fig. 5E, Bob has granted Alice authorization to write 530e, and Carl has granted Alice authorization to read, but only if Bob has also granted Alice authorization to read 532e. As shown in Fig. 5E, the fixpoint for this set of certificates is an authorization map in which Alice has been given authorization to write from Bob, but has no other authorizations. Similarly, in Fig. 5F Bob has granted Alice authorization to read and write 530f, while Carl has granted Alice authorization to read to the extent that Bob has

also authorized Alice to read 532f. When the least fixpoint is computed 502f, Alice will have authorization from Bob to read and write, and authorization from Carl to read.

[056] Fig. 5G illustrates a chain of delegation. In Fig. 5G, the trust management engine is presented with three certificates. Certificate 530g indicates that Bob has authorized Alice to write. Certificate 532g indicates that Carl has authorized Alice to do whatever Bob has authorized her to do. And certificate 534g indicates that Dave has authorized Alice to do whatever Carl has authorized her to do. As shown in Fig. 5G and Fig. 6, the least fixpoint can be computed as follows. First, the trust management engine examines the certificates to identify the relevant authorizing principals – in this case, Bob, Carl, and Dave (610). Next, the state of the first authorization map (i.e., row 1 of table 502g) is initialized to "No" (612). The trust management engine then evaluates each of the certificates using the authorization information contained in row 1 of table 502g (614), and fills in row 2 of the table with the results (616). On the trust management engine's first pass through the certificates (613 – 618), only Bob's authorization state will change. Carl and Dave's authorization states remain unchanged, since certificates 532g and 534g both evaluate to "No" using the information contained in row 1 of table 502g (for example, certificate 532g evaluates to "No" because it only authorizes whatever Bob has authorized, and in row 1, Bob has not authorized anything). Next, row 2 is compared with row 1 (620). If they are the same (i.e., a "Yes" exit from block 620), then a fixpoint has been reached and further evaluation of the certificates is unnecessary. However, where, as here, row 1 is not the same as row 2 (i.e., a "No" exit from block 620), the certificates are evaluated again, this time using the information in the most-recently updated row – i.e., row 2 (613 – 620). As shown in Fig. 5G, on the trust management engine's second pass through the certificates, Carl's state is changed from N to W, and on the third pass through the certificates Dave's state is changed from N to W. If a fourth pass were conducted, the state of each of the principals would remain unchanged (i.e., Bob, Carl, and Dave would still indicate that only write access was authorized), and the certificate evaluation process would end (622).

[057] Fig. 5H illustrates the process of making a trust management decision where the certificates include a multi-way delegation. In Fig. 5H, certificate 530h

indicates that Bob has authorized Alice to write, certificate 532h indicates that Carl has authorized Alice to read and write, and certificate 534h indicates that Dave has authorized Alice to do whatever both Bob and Carl have authorized her to do. As seen in Fig. 5H, the process of computing the fixpoint remains the same as previously described. On the first pass through the certificates, the table is updated to reflect the direct authorizations from Bob and Carl (see row 2 of table 502h). On the second pass through the certificates, the table is updated to show that Dave has authorized write access (that is, Dave has authorized Alice to do whatever *both* Bob and Carl have authorized her to do – i.e., write). A third pass would not result in any additional state changes, thus indicating that a fixpoint had been reached on the previous pass.

[058] Fig. 5I illustrates the process of making a trust management decision where the certificates include an inter-assertion communication. In Fig. 5I, certificate 530i indicates that Bob has authorized Alice to write, certificate 532i indicates that Bob has also authorized Alice to do whatever Carl has authorized her to do, and certificate 534i indicates that Carl has authorized Alice to read, but only if Bob has authorized her to write. As seen in Fig. 5I, computation of the fixpoint is straightforward. On the first pass through the certificates, table 502i is updated to reflect the direct authorization from Bob that is expressed in certificate 530i (see row 2). On the next pass through the certificates, table 502i is updated to show that Carl has authorized read access (since Bob has authorized write access). Finally, on the third pass through the certificates, Bob's state is updated to show that he has granted read and write access, since certificate 532i indicates that Bob will authorize whatever Carl authorizes, and Carl has authorized read access.

[059] Figs. 5A-5I and 6 thus illustrate the operation of a basic trust management engine in accordance with an embodiment of the present invention. Figs. 7-10 illustrate various optimizations and alternative embodiments.

[060] Referring to Fig. 7, a process is shown for making trust management decisions. The process shown in Fig. 7 is closely related to the process shown in Fig. 6, but may be more efficient in certain situations. The difference between Fig. 7 and Fig. 6 is the addition of block 702 between blocks 616 and 618. Block 702 checks the authorization state of the principal from whom authorization is sought (i.e., the root

principal), and terminates further processing of the certificates if authorization is detected (i.e., a "Yes" exit from block 702). If authorization is not detected, then processing continues in the manner described above in connection with Fig. 6. The optimization shown in Fig. 7 is useful when, for example, the trust management engine really only cares about the authorization of one principal. Referring to Fig. 5I, for example, if the trust management engine only needed to know whether Carl had authorized Alice to read, it could stop processing the certificates as soon as this condition was detected during the second processing pass (i.e., eliminating the remaining portion of the second processing pass, and the entire third and fourth passes). This follows from the fact that the certificates are monotonically increasing functions, and thus, once an authorization is granted, subsequent processing of the certificates will not indicate that the authorization should be denied (conversely, if the authorizations were expressed as monotone decreasing functions, once a denial of authorization was identified, further processing would not identify an authorization grant). Similarly, if the trust management engine needed to determine whether Bob had granted Alice authorization to read, further processing of the certificates could be terminated when this condition was detected during the third processing pass (i.e., eliminating the remaining portion of the third processing pass and the need to perform a fourth processing pass to confirm that a fixpoint had been reached).

[061] Note, however, that while Fig. 7 shows block 702 appearing between blocks 616 and 618, it should be appreciated that block 702 could be placed at various other points in the process. For example, block 702 could be placed between blocks 618 and 620 (i.e., to perform an inter-processing pass check instead of an intra-pass check), or between 620 and 613. Thus one of ordinary skill in the art should appreciate that the order of the blocks shown in Fig. 7 (and in the other process flow diagrams appearing herein) can be varied without departing from the principles of the present invention.

[062] Fig. 8 illustrates another optimization of the process shown in Fig. 6. The process shown in Fig. 8 differs from that shown in Figs. 6 and 7 in that only one row of the authorization map is maintained, and this row is updated on-the-fly as each certificate is evaluated (blocks 802 and 804). Thus, referring to Fig. 5G for example, the fixpoint

could theoretically be reached in one pass, as opposed to three, if the certificates were evaluated in the right order. In particular, if certificate 530g is evaluated first, the state of the authorization map will be changed immediately to indicate that Bob has given Alice permission to write. If certificate 532g is evaluated next, it will indicate that Carl has given Alice permission to write, and this information will also be immediately included in the authorization map. Thus, when certificate 534g is evaluated, it will conclude that Dave has given Alice permission to write. When the authorization map is updated to reflect Dave's authorization, the authorization map will be identical to row 4 of table 502g in Fig. 5G (i.e., W W W), but, in contrast to the map shown in Fig. 5G, this authorization map will have been generated by evaluating three certificates as opposed to nine (i.e., three passes through the three certificates). Of course, if the certificates were evaluated in a different order, more than three certificate evaluations may be needed. Thus, in embodiments where the certificates are processed in an arbitrary order, the procedure shown in Fig. 8 can be expected (but not guaranteed) to result in some level of processing efficiency.

[063] As shown in Fig. 8, if at some point all of the certificates are evaluated without any change being entered into the authorization map (i.e., a "No" exit from block 806), then a fixpoint has been reached and further processing can be terminated. Otherwise, processing of the certificates continues until a fixpoint is reached or until some desired authorization is detected (e.g., a "Yes" exit from block 702). Thus, referring to Fig. 5G, for example, further processing of the certificates can be terminated any time all three certificates are evaluated without any change in the authorization map. In the context of Fig. 5G, this might occur if the trust management engine attempted to determine whether any principal had authorized Alice to read. Since, in Fig. 5G, Alice does not have this authorization, a "Yes" exit from block 702 would not occur, and the process would eventually terminate via a "No" exit from block 806.

[064] Fig. 9 shows yet another technique for making trust management decisions. The process shown in Fig. 9 differs from the processes shown in Figs. 6-8 in that authorization tables like those shown in Figs. 5A-5I are not maintained. Instead, a graph is constructed that reflects the dependencies (if any) between the various

principals. The certificates are then iteratively evaluated, and the authorization states of the principals updated, until a desired authorization is detected or until a steady-state/fixed point is reached.

[065] Referring to Fig. 9, the trust management engine first obtains a set of certificates and identifies the relevant principals (902). Next, the trust management engine forms a dependency graph by assigning a node to each principal (904), and connecting these nodes according to the relationships specified in the certificates (906).

[066] Once the dependency graph has been generated, each of the direct authorizations in the group of certificates is evaluated (908), and if the state of any of the nodes changes as a result, that node is entered onto a list for further processing. In one embodiment, the list is implemented as a first-in, first-out queue. For each node on the list (910), each one of the certificates that depend on the state of the listed node is evaluated (912, 914), and if the state of any other node changes as a result (i.e., a "Yes" exit from block 916), that node's state is updated (918) and the node is added to the list (unless it is already there)(920). A fixpoint is reached when there are no more nodes on the list. Alternatively, processing could be terminated before the fixpoint was reached if a desired level of authorization was detected (e.g., if a predefined node's authorization increased to a certain level).

[067] Fig. 10 shows several illustrative dependency graphs corresponding to the sets of certificates discussed above in connection with Figs. 5G, 5H, and 5I. Referring to Fig. 10, graph 1002g includes three nodes – B, C, and D – corresponding to the three principals discussed in connection with Fig. 5G (i.e., Bob, Carl, and Dave, respectively). Node B issues one certificate, represented by arrow 530g. As shown in Fig. 5G, this certificate indicates that Bob authorizes Alice to write. Since this certificate is not dependent on the state of any of the other nodes, arrow 530g does not connect to any other node. As shown in Fig. 5G, Carl (represented in Fig. 10 by node C) issues a certificate indicating that Alice may do whatever Bob says she may do. Since evaluation of this certificate depends on the state of node B (i.e., Bob), an arrow 532g is drawn from node B to node C. If Bob's state changes, then certificate 532g will be evaluated, and Carl's state updated if necessary. Similarly, an arrow 534g is drawn between node C and

node D to represent the dependence between Carl's state and Dave's state via certificate 534g.

[068] Graph 1002h shows the dependency graph for the set of certificates 530h, 532h, 534h shown in Fig. 5H. As shown in Fig. 5H and Fig. 10, certificates 530h and 532h are direct authorizations, and thus do not connect to other nodes. Certificate 534h, on the other hand, expresses a dependency between Dave's state (node D), and the states of Bob and Carl (nodes B and C). When the process shown in Fig. 9 is carried out, nodes B and C will initially be added to the node list (908), since evaluation of certificates 530h and 532h will change their respective states. When processing of the list reaches either of these nodes (910), certificate 534h will be evaluated (914), and the state of node D updated (916, 918). Although Fig. 9 would imply that node D should be added to the list, since no other nodes are dependent on the state of node D, this would be unnecessary. Thus, in some embodiments the trust management engine might decline to add nodes to the list that do not have other nodes that depend directly or indirectly on their state.

[069] Similarly, Graph 1002i shows the dependency graph for the set of certificates 530i, 532i, 534i shown in Fig. 5I. As shown in Fig. 5I and Fig. 10, certificate 530i is a direct authorization, while certificates 532i and 534i express the dependency of Bob's state on Carl's state and vice-versa. When the process shown in Fig. 9 is carried out, direct authorization 530i will be evaluated and node B will be added to the node list (908). When node B is processed (910, 912), certificate 534i will be evaluated (914), the state of node C will be updated (916, 918), and node C will be added to the list (920). Similarly, when node C is processed, certificate 532i will be evaluated, the state of node B will be updated, and node B will be added to the list. When node B is processed again, the state of node C will not change when certificate 534i is evaluated, and the fixpoint computation process will terminate (i.e., a "No" exit from block 924).

[070] One of ordinary skill in the art will appreciate that the processes and structures illustrated in Figs. 5A - 10 can be implemented in any of a variety of ways, including via well-known programming techniques and standard programming languages such as C, C++, Java, Lisp, Perl, ML, or the like. For example, the dependency graphs of Figs. 9 and 10 could be implemented using an adjacency list representation of the graph,

and a worklist algorithm could be used to compute the fixpoint. Worklist algorithms are well-known in the art, and examples can be found in, e.g., Muchnick, *Advanced Compiler Design and Implementation*, pages 232-233 (Morgan Kaufman 1997), which is hereby incorporated by reference. In one embodiment, each node in the dependency graph is assigned its own adjacency list. Entries in the adjacency list correspond to entries/nodes in the graph, and may specify the various successor nodes and the certificates/functions corresponding thereto.

[071] It will be appreciated that other modifications could be made to the processes shown in Figs. 6-10 without departing from the principles of the present invention. For example, in the processes shown in Figs. 6-10, certificates that express direct authorization (such as certificate 530g in Fig. 5G) need not be evaluated more than once, since further evaluation will not have an effect on the authorization map/graph.

[072] Embodiments of the present invention that make use of SPKI certificates will now be described. The Simple Public Key Infrastructure (SPKI) is a well-known framework for expressing authorizations. SPKI certificates can contain two types of assertions – *name assertions* and *authorization assertions*. The SPKI authorization lattice can be viewed as a powerset lattice with two kinds of authorizations, *name* and *action*. The meaning of a name is taken relative to a given principal. A *full name* is a principal and a sequence of names. A name assertion is represented by a 4-tuple $\langle p,n,s,\langle t_1,t_2\rangle\rangle$, which means that principal, p, authorizes subject s to act on behalf of name n at any time between t_1 and t_2. The subject field can either directly identify a full name, or specify a threshold subject of the form, $\langle k,\{f_1,f_2,\ldots\}\rangle$. A principal can only act for a threshold subject if at least k of the names $f_1, f_2,\ldots$ denote that principal.

[073] An authorization assertion is represented by a 5-tuple $\langle p,s,d,x,\langle t_1,t_2\rangle\rangle$ which means that principal p authorizes subject s to perform (and delegate, if d is true) the operations allowed by action x at any time between t_1 and t_2. In general terms, actions denote sets of s-expressions, and can comprise virtually any operation one might want to control with a trust management system.

[074] The process of making an SPKI trust management computation will now be described. Suppose it is desired to know if principal p authorizes principal p' to

perform a certain operation at time t using the assertions A. To make this determination, a trust management engine evaluates the certificates for a least fixpoint, or for an approximation of the least fixpoint that is sufficient to show that p' is authorized to perform the operation. Examples of such a computation are provided below in Figs. 12 and 13, and additional information regarding the process of making SPKI trust management decisions in accordance with the present invention can be found in Appendix A.

[075] Note that, in general, name assertions affect the meaning of authorization assertions, but not vice-versa. Thus, instead of computing a least fixpoint over the entire SPKI authorization map lattice for a given trust management decision, it is possible to first compute a fixpoint over the principal-name lattice (i.e., to resolve the name assertions), and then compute a fixpoint over the principal-authorization lattice. The first fixpoint computes for each principal, p, the pairs $<p', n>$ such that p authorizes p' to act on behalf of name n at time t. The second fixpoint computes for each principal p whether or not the request u is authorized by p. If there are c assertions, the first fixpoint will generally require $O(c^2)$ space and the second only $O(c)$ space. In practice, the time for trust management computation will often be small in comparison to the cryptographic costs of signature checking (in some embodiments, however, signature checking may only be performed once, before the verified certificates are placed in secure storage). Appendix A – and section 3 and Figs. 17 and 18 in particular – provides more information on how to express SPKI in the framework presented herein.

[076] Fig. 11 shows another example of a process that can be governed using a trust management engine operating in accordance with the present invention. Fig. 11 illustrates a scenario in which a user attempts to move a piece of protected content from one device to another. For example, the user may attempt to download a digitally-encoded song 1106 from a kiosk 1104 to an electronic music player 1100, or may attempt to download a digitally-encoded movie from an Internet website. As shown in Fig. 11, the user (and/or the kiosk) may have one or more SPKI certificates 1102 issued by other entities in the system. For example, the user or kiosk may have a certificate from the music producer, one or more distributors, a rights management provider, and/or the like.

These certificates may have been obtained through any of a variety of means. For example, the user may have downloaded some of these certificates in connection with purchasing the content, others in connection with purchasing the application that renders the content, and others when installing the rights management software/hardware that controls the use of the content.

[077] As shown in Fig. 11, a user indicates that he or she wishes to move content item 1106 from device 1104 to device 1100. The trust management system 1108 on kiosk 1104 receives this request 1110 (and possibly additional certificates 1102) and determines whether the user has authorization to perform the requested action. To make this determination, the trust management engine 1108 uses SPKI certificates 1102. As previously described, the trust management engine takes the certificates and determines whether they indicate that the user has authorization to perform the requested action. For example, to download a song to a portable device from an in-store kiosk, the user may need the authorization of the content owner, the store, and/or the kiosk provider. While the kiosk may be able to grant its authorization to the user directly, the content owner's authorization may, for example, be deducible only by tracing back through the authorizations obtained from various other entities in the song's distribution chain.

[078] Moreover, by presenting the appropriate certificates, it may be possible for the portable device to establish authorization to download the content even though no direct authorization was obtained from the kiosk (e.g., if the user had previously purchased the content from an affiliate distributor, and the kiosk was authorized to download content if authorized by the affiliate). Here, the kiosk's trust management system might still wish to verify that the user has authorization to download the content, but will need to examine the chain of certificates presented by the user and/or stored in, or obtained by, the kiosk. Alternatively, or in addition, the device (or a trust management engine thereon) might wish to verify the kiosk's authorization to store and supply content.

[079] If a trust management engine determines that the user and/or the kiosk does not have proper authorization, then the download is prevented (or, if the download has already occurred, a rights management engine on the user's system may simply deny

access to the content); otherwise, the action is performed in accordance with the user's request. In some embodiments the authorization process may not be a simple yes/no decision, and different consequences might flow from the level of authorization that was detected. In addition, the response to a "no" determination may not be to prevent the action, but may instead be to take some other action, such as sending a notification to some entity responsible for monitoring such occurrences.

[080] Figs. 12-13 illustrate the operation of a trust management engine that processes SPKI certificates and renders trust management decisions. Specifically, Figs. 12 and 13 show a set of SPKI assertions from which a trust management engine will attempt to determine whether a content owner has authorized a particular device to download a particular content item. Table 1202 contains a set of SPKI name assertions, and table 1302 contains a set of SPKI authorization assertions. Tables 1204 and 1304 illustrate how to use these certificates to make efficient trust management decisions.

[081] Referring to Figs. 12 and 13, given a request by specific device ("Device No. 123") to download protected content from a Retail Store, a trust management engine processes the certificates shown in Figs. 12 and 13 to determine if the request should be granted. For example, the trust management engine might frame the question as: Is Device No. 123 authorized by the owner of the requested content (i.e., "Music Co.") to download it? To arrive at this question, the trust management engine (or a rights management engine of which it is a part) may, for example, have consulted a database indicating that authorization from Music Co. is required before the particular content item can be downloaded. This could be determined by, e.g., examining the terms of the distribution agreement between Music Co. and the Retail Store, or an electronic representation thereof maintained by the trust/rights management engine.

[082] Since the authorization certificates shown in table 1302 do not explicitly refer to Device No. 123, the trust management engine might first determine whether Device No. 123 is a Retail Store Supported Device. If the answer is "yes", then the trust management engine could determine whether Device No. 123, as a Retail Store Supported Device, is authorized by Music Co. to download the requested content. Alternatively, the order of these two decisions could be switched. That is, the trust

management engine might first determine if a Retail Store Supported Device is authorized by Music Co. to download the requested content, and then determine whether Device No. 123 is a Supported Device. In other embodiments, these determinations can be made concurrently or substantially concurrently.

[083] Fig. 12 shows a set of SPKI name assertions that can be used to complete the processing of the request described above. For clarity, the SPKI certificate validity period is not shown. To determine whether Device No. 123 is a Retail Store Supported Device, the trust management engine processes the certificates shown in table 1202. Table 1204 illustrates a technique for making this determination. In particular, table 1204 shows a least fixpoint solution to the question: What set of principals does each issuer's name denote? As seen in row 4 of table 1204, Device No. 123 is a Retail Store Supported Device. Having determined that Device No. 123 is a Supported Device, the trust management engine now attempts to determine whether Device No. 123, as a Supported Device, has authorization from Music Co. to download the requested content.

[084] Referring to Fig. 13, table 1302 shows a set of SPKI authorization assertions. For ease of explanation, the SPKI certificate validity period has been omitted. Table 1304 illustrates a technique for using the certificates contained in table 1302 to answer the question: Has Music Co. authorized Device No. 123 to download the requested content? As seen in row 4 of table 1304, Music Co. has authorized Device No. 123 to download the requested content item. (Note that in evaluating the Retail Store's certificate, the result of the first fixpoint tells us that Device No. 123 is a Supported Device.)

[085] Thus, it should be appreciated that the systems and methods of the present invention can be used to implement trust management engines for a wide variety of trust management systems. For example without limitation the framework of the present invention can be applied to trust management systems such as Keynote (as discussed in Appendix A) in addition to SPKI and the trust management system discussed in connection with Figs. 5A – 5I.

[086] As described above, the problem of access control can be broken into two sub-problems: determining whether or not a request should be allowed, and enforcing the

decision. The first sub-problem can be solved by a trust management system that uses the techniques described above to determine when requests are authorized. The second sub-problem – enforcing the decision – can be solved by applying a suitable combination of security techniques and principles, such as encryption, tamper resistance, physical protection, security-by-obscurity, or the like. Several security considerations are discussed below; it should be appreciated, however, that for purposes of practicing some embodiments of the present invention, any suitable security techniques can be used. In some embodiments, no special security measures are taken beyond that which is inherent in the trust management decision process.

[087] The effectiveness of the trust management system will generally depend, at least in part, on the integrity and authenticity of the certificates that are used to make the trust management decisions. If these certificates have been improperly modified, and/or have been forged, the trust management system may make decisions that are contrary to the intentions of the authorizing principals. A wide variety of cryptographic and other techniques exist which can be used to protect the integrity and authenticity of the certificates, and/or to revoke certificates. For example, the present invention can be used in connection with the security mechanisms described in Menezes at pages 425-586 and 645-662, or in commonly-assigned U.S. Patent 6,157,721, issued Dec. 5, 2000, entitled "Systems and Methods Using Cryptography to Protect Secure Computing Environments," and U.S. Patent Application No. 09/628,692, entitled "Systems and Methods for Using Cryptography to Protect Secure and Insecure Computing Environments," each of which is hereby incorporated by reference. Alternatively, or in addition, security can be achieved through physical means, such as, for example, isolating the computer systems on which the trust management engine runs. For example, the trust management engine might be implemented on an enterprise server that is password-protected and locked in a secure room. The certificates needed to make trust management decisions might be generated by processes running on the server, or by systems or processes that are in secure communication therewith.

[088] In some embodiments the trust management engine operates in connection with, and/or forms part of, a larger rights management or enforcement engine that is

responsible for controlling and/or monitoring access to electronic content and other resources in accordance with certain rules and policies. Additional information on rights management systems can be found in commonly-assigned U.S. Patent No. 5,892,900, entitled "Systems and Methods for Secure Transaction Management and Electronic Rights Protection", issued April 6, 1999 ("the '900 patent"), which is hereby incorporated by reference in its entirety. For purposes of practicing the present invention, any suitable rights management engine could be used, including without limitation embodiments of the Rights Operating System software described in the '900 patent, the InterRights Point™ software or Rights/System™ software developed by InterTrust Technologies Corporation of Santa Clara, California, or any of the other commercially-available digital rights management technologies. In other embodiments, the trust management engine does not form part of a larger digital rights management product or process.

[089] In a preferred embodiment, the trust management engine is protected from tampering and/or other improper access. This can be accomplished using a combination of tamper resistance techniques and cryptographic protections, such as those described in the '900 patent or in commonly-assigned U.S. Patent Application No. 09/095,346, entitled "Obfuscation Techniques for Enhancing Software Security," filed June 9, 1998 ("the '346 application"), which is hereby incorporated by reference in its entirety. If the trust management engine forms part of a larger rights management engine, the mechanisms used to protect the rights management engine will typically serve to protect the trust management engine as well. Alternatively, or in addition, aspects of the trust management engine can be implemented in (and/or designed to take advantage of) hardware security mechanisms such as secure processing units, protected memory, and the like.

[090] Fig. 14 shows an example of a computer system 1400 that can be used to practice embodiments of the present invention. Computer system 1400 may comprise a general-purpose computing device such as a personal computer or network server, or a specialized computing device such as a cellular telephone, personal digital assistant, portable audio or video player, television set-top box, kiosk, or the like. Computing device 1400 will typically include a processor 1402, memory 1404, a user interface 1406,

a port 1407 for accepting removable memory 1408, a network interface 1410, and a bus 1412 for connecting the aforementioned elements. The operation of computing device 1400 will typically be controlled by processor 1402 operating under the guidance of programs stored in memory 1404. Memory 1404 will generally include both high-speed random-access memory (RAM) and non-volatile memory such as a magnetic disk and/or flash EEPROM. Some portions of memory 1404 may be restricted, such that they cannot be read from or written to by other components of the computing device 1400. Port 1407 may comprise a disk drive or memory slot for accepting computer-readable media such as floppy diskettes, CD-ROMs, DVDs, memory cards, other magnetic or optical media, or the like. Network interface 1410 is typically operable to provide a connection between computing device 1400 and other computing devices (and/or networks of computing devices) via a network 1420 such as the Internet or an intranet (e.g., a LAN, WAN, VPN, etc.). In one embodiment, computing device 1400 includes a secure processing unit 1403 such as that described in the '900 patent. A secure processing unit can help enhance the security of sensitive operations such as key management, signature verification, and other aspects of the trust management decision and enforcement process.

[091] As shown in Fig. 14, memory 1404 of computing device 1400 may include a variety of programs or modules for controlling the operation of computing device 1400. For example, memory 1404 preferably includes a program 1430 for implementing some or all of the functionality of the trust management engine described above. In some embodiments, the trust management engine may also be capable of applying policies, rules, and/or controls to govern the use of content or the performance of events, and/or these policies, rules, and/or controls may be applied by a rights management program 1429 such as that described in the '900 patent. Memory 1404 may also include a program – such as that described in U.S. Patent Application No. 09/617,148, entitled "Trusted Storage Systems and Methods", filed July 17, 2000, which is hereby incorporated by reference – for maintaining a database of protected data such as cryptographic keys, certificates, or the like. In addition, memory 1404 may contain protected content 1434.

[092] One of ordinary skill in the art will appreciate that the systems and methods of the present invention can be practiced with computing devices similar or identical to that illustrated in Fig. 14, or with virtually any other suitable computing device, including computing devices that do not possess some of the components shown in Fig. 14 and/or computing devices that possess other components that are not shown. Thus it should be appreciated that Fig. 14 is provided for purposes of illustration and not limitation as to the scope of the invention.

APPENDIX A: MATHEMATICAL BACKGROUND & ADDITIONAL EMBODIMENTS

The following discussion provides a mathematical framework for certain embodiments of the present invention. It should be appreciated, however, that some embodiments of the present invention may be practiced without conforming to all of the mathematical "requirements" and optimizations set forth below.

1 Introduction

This paper presents a mathematical framework for expressing trust management systems. The framework can express many well-known systems, including KeyNote [2] and SPKI [8]. Trust management systems can be concisely specificied, which helps in comparing current systems and in analyzing the design tradeoffs of new systems. The framework defines the semantics of a trust management engine as a least fixpoint in a lattice, which for many instantiations directly leads to an implementation. In the case of SPKI, the paper shows how an implementation of the semantics can make a trust management decision more simply and efficiently than the combination of certificate path discovery [5, 6] and tuple reduction [8].

Section 2 defines the framework and presents an instantiation via an extended example of a toy trust management system. Sections 3 and 4 show how to express SPKI and Keynote as instantiations of the framework. Section 5 formalizes several concepts generally applicable to trust management systems. Section 6 reviews mathematical background and defines notation. Readers unfamiliar with lambda expressions or lattices should refer to the appendix as necessary.

2 Framework

This section defines the framework for expressing trust management systems and presents an extended example. The idea behind the framework is to leave open the kinds of authorizations that can be made by the system, only requiring that they satisfy a few natural mathematical properties. The framework defines the kinds of assertions that can be made by entities in the system and gives a precise semantics specifying what a collection of assertions means and when a request should be granted. In many cases, a direct implementation of this semantics leads to a viable trust management engine.

The elements of the framework are principals, authorizations, authorization maps (abbreviated authmaps), licenses, and assertions. A *principal* is an atomic entity that may make or authorize requests. We use p to range over principals.

$$p \in \mathit{Principal}$$

For the framework, the only necessary property of principals is that they are distinguishable. In a real implementation, for cryptographic reasons, a principal might correspond to a public key. In the example of this section, we will deal with the principals Alice, Bob, Carl, and Dave.

An *authorization* expresses the permissions granted by a principal. Authorizations form a lattice, *Auth*, where $u \sqsubseteq u'$ means that u' permits more operations than u.

$$u \in \mathit{Auth}$$

For a given trust management decision, each principal in the system will be associated with a single authorization that describes the permissions granted by that principal. The least upper bound operator $\bigsqcup$ defines how to sensibly combine multiple authorizations made by the same principal into a single authorization. The *Auth* lattice can be instantiated in varying ways to express different trust management systems.

As an example, imagine that *Auth* specifies whether or not Alice may read some particular file, write that file, or both. Let R denote that Alice may read the file, W denote she may write the file, RW denote she may read and write the file, and N denote she may do neither. So, $\mathit{Auth} = \{\mathsf{N}, \mathsf{R}, \mathsf{W}, \mathsf{RW}\}$. In order to make *Auth* into a lattice, define $\mathsf{N} \sqsubseteq \mathsf{R}$, $\mathsf{N} \sqsubseteq \mathsf{W}$, $\mathsf{R} \sqsubseteq \mathsf{RW}$, $\mathsf{W} \sqsubseteq \mathsf{RW}$ and $\bigsqcup\{\mathsf{R}, \mathsf{W}\} = \mathsf{RW}$. Note that this authorization lattice only specifies the authorizations granted to Alice for a particular file, not for any other principal or any other file. To represent such authorizations, we could use a more complicated lattice like $\mathit{Principal} \times \mathit{File} \longrightarrow \{\mathsf{N}, \mathsf{R}, \mathsf{W}, \mathsf{RW}\}$, but we will stick with the simple lattice as the example for this section.

An *authmap* is a function mapping principals to authorizations that describes the authorizations made by each principal in a trust management decision.

$$m \in \mathit{AuthMap} = \mathit{Principal} \longrightarrow \mathit{Auth}$$

Recall from Section 6 that *AuthMap* is a lattice under the pointwise ordering because *Auth* is a lattice. For our example lattice, an authmap expresses the authorization each principal grants to Alice to read or write the file. Such a map might be m, where $m(\mathsf{Bob}) = \mathsf{R}$ and $m(\mathsf{Carl}) = \mathsf{RW}$.

A *license* grants authorization, expressed as a monotone function from authmaps to authorizations.

$$l \in \mathit{License} = \mathit{AuthMap} \longrightarrow_m \mathit{Auth}$$

Intuitively, the meaning of license l is that if principals grant authorizations as given by authmap m, then l grants the authorization $l(m)$. Figure 15 shows some licenses for the example lattice. License 2, which delegates to Bob, shows how

the dependence of licenses on authmaps expresses delegation. The monotonicity requirement means that the authorizations granted by a license can only increase as a consequence of an increase in other principals' authorizations. The reader should verify that all of the licenses in Figure 15 are monotone. In particular, in licenses 6, 7, and 8, note that if $=$ were used instead of $\sqsubseteq$, the license would not be monotone. Monotonicity is required in order to give a sensible meaning to a collection of assertions so that the trust management engine will be well-defined.

An *assertion* is an expression of authorization made by a principal. Assertions are the framework's abstraction of digital certificates. Formally, an assertion consists of a principal and a license.

$$a \in \mathit{Assertion} = \mathit{Principal} \times \mathit{License}$$

Assertion $\langle p, l \rangle$ should be read as "p authorizes l". Principal p is referred to as the *issuer*. Any principal could issue an assertion with any of the licenses from Figure 15. In a real implementation, assertions would typically be signed by the public key of the issuing principal and would require signature checking before being processed by the trust management engine.

A trust management engine must take a set of assertions made by various principals, some of whom may delegate to each other, and find a coherent authmap representing the authorizations of those principals. The assertions may reside in the local machine, or may be presented by the principal making the request, and may have been created in various places. The act of finding a coherent authmap from various assertions is the component that distinguishes trust management from traditional access control mechanisms. We define the semantics of a set of assertions as follows (as we do throughout the document, we use $\mathcal{M}$ with a subscript to denote a semantic function).

$$\mathcal{M}_{\mathit{Assertions}} : \mathcal{P}(\mathit{Assertion}) \longrightarrow_m \mathit{AuthMap}$$
$$\mathcal{M}_{\mathit{Assertions}}(A) = \mathit{lfp}(\lambda m.\lambda p. \bigsqcup \{l(m) \mid \langle p, l \rangle \in A\})$$

The intuition behind the definition of $\mathcal{M}_{\mathit{Assertions}}$ is that it combines all of the assertions issued by each principal into one authorization for that principal, taking delegation into account. In more detail, the set of assertions made by a principal p is $\{l \mid \langle p, l \rangle \in A\}$. Given the authorization map m, the set of authorizations granted by p is $\{l(m) \mid \langle p, l \rangle \in A\}$. By taking a least upper bound, we can combine the authorizations into a single authorization granted by p, namely $\bigsqcup \{l(m) \mid \langle p, l \rangle \in A\}$. Finally, to find an authorization map that is consistent with all of the licenses, we take a least fixpoint in the *AuthMap* lattice, which relies on the fact that licenses are monotone. Because the licenses in assertions are monotone, so is $\mathcal{M}_{\mathit{Assertions}}$. That is, if more assertions are input to $\mathcal{M}_{\mathit{Assertions}}$, then more authorizations will be made by the resulting authmap. The definition of $\mathcal{M}_{\mathit{Assertions}}$ as a least fixpoint also makes it clear why the framework cannot handle revocation, since that would require non-monotone assertions.

Figure 16 shows some example computations of $\mathcal{M}_{\mathit{Assertions}}$. Each row has a set of assertions in the left column, a fixpoint computation of an authmap in the middle, and a comment about the example in the right column. The order of assertions in the left column is irrelevant. The fixpoint computation (see Section 6 for an explanation) is shown as a sequence of authmaps, one per row, where each column contains the authorizations made by a principal. Each row contains an authmap consisting of the least upper bound of all of the assertions applied to the authmap in the previous row. The final row is the least fixpoint authmap of the set of assertions. The examples give some idea of how least fixpoint computation can express concepts like path validation [13], chaining and five-tuple reduction [8], and inter-assertion communication [4].

A trust management engine makes a decision based on a request, an authorizing principal, and a set of assertions. The authorizing principal reflects the policy governing the request. A request is expressed as an element of the *Auth* lattice. The semantics is defined by $\mathcal{M}_{\mathit{Engine}}$, where $\mathcal{M}_{\mathit{Engine}}(p, u, A)$ means that principal p authorizes request u according to the assertions in A.

$$\mathcal{M}_{\mathit{Engine}} : \mathit{Principal} \times \mathit{Auth} \times \mathcal{P}(\mathit{Assertion}) \longrightarrow \mathit{Bool}$$
$$\mathcal{M}_{\mathit{Engine}}(p, u, A) = u \sqsubseteq \mathcal{M}_{\mathit{Assertions}}(A)(p)$$

The trust engine computes the authmap corresponding to the provided assertions A and determines if the request requires less authorization than is granted by the authorizing principal p. For the example lattice, if Alice would like to write the file (which is denoted by the request W) under Bob's control and the assertions a_1, a_2 and a_3 are available, the trust engine would compute $\mathcal{M}_{\mathit{Engine}}(\mathsf{Bob}, \mathsf{W}, \{a_1, a_2, a_3\})$ to determine if Bob authorizes Alice to write.

The trust management engine does not necessarily need to compute the entire fixpoint as defined by $\mathcal{M}_{\mathit{Assertions}}$. To prove that a request is justified, all the engine must find is an authmap m such that $u \sqsubseteq m \sqsubseteq \mathcal{M}_{\mathit{Assertions}}(A)(p)$. Finding such an m may be easier than computing the fixpoint for two reasons. First, the trust engine may be able to work in a lattice in which elements are more compactly represented and licenses are more efficiently computed. Second, the engine may terminate the

fixpoint computation early, once a large enough m has been found. The sections on KeyNote and SPKI will take advantage of this fact.

Figure 17 summarizes the entire framework. An instantiation of the framework defines a trust management system by providing *Auth* lattice, a language for expressing licenses, and a means of computing $\mathcal{M}_{Engine}$. The art in designing a trust management system lies in choosing a lattice of authorizations and a subset of the monotone functions to be used as licenses. This controls the expressiveness of the system. The engineering comes in choosing a representation for authorizations and a language for licenses so that the desired monotone functions can be expressed concisely and so that $\mathcal{M}_{Engine}$ can be computed efficiently. The rest of this section is devoted to a toy language for expressing licenses in the example lattice of this section. The following sections will show how to express more realistic trust management systems.

2.1 A toy trust management language

Using all of mathematics to write licenses is fine for expository purposes, but in order to build a practical trust management engine, one must define a language for expressing licenses and give a way for the trust management engine to compute the authorization expressed by a license. Here is an example language of expressions for $Auth = \{\mathsf{N}, \mathsf{R}, \mathsf{W}, \mathsf{RW}\}$.

$$
\begin{array}{rcl}
e & ::= & \texttt{N} \mid \texttt{R} \mid \texttt{W} \mid \texttt{RW} \\
& \mid & p \\
& \mid & \texttt{(glb } e \ldots\texttt{)} \\
& \mid & \texttt{(lub } e \ldots\texttt{)}
\end{array}
$$

A license expression is either a constant denoting the corresponding lattice element, or a principal (in unspecified format) denoting delegation to a principal, or the greatest lower bound (`glb`), or the least upper bound (`lub`) of a sequence of other expressions. Let *ExampleLicense* be the set of expressions generated by the above grammar. We can give a semantics to expressions by defining a function $\mathcal{M}_{EL}$ inductively on the structure of expressions.

$$\mathcal{M}_{EL} : \mathit{ExampleLicense} \longrightarrow \mathit{License}$$

$$
\begin{array}{rcl}
\mathcal{M}_{EL}(\texttt{N}) & = & \lambda m.\, \mathsf{N} \\
\mathcal{M}_{EL}(\texttt{R}) & = & \lambda m.\, \mathsf{R} \\
\mathcal{M}_{EL}(\texttt{W}) & = & \lambda m.\, \mathsf{W} \\
\mathcal{M}_{EL}(\texttt{RW}) & = & \lambda m.\, \mathsf{RW} \\
\mathcal{M}_{EL}(p) & = & \lambda m.\, m(p) \\
\mathcal{M}_{EL}(\texttt{(glb } e \ldots\texttt{)}) & = & \lambda m.\, \bigsqcap\{\mathcal{M}_{EL}(e)(m), \ldots\} \\
\mathcal{M}_{EL}(\texttt{(lub } e \ldots\texttt{)}) & = & \lambda m.\, \bigsqcup\{\mathcal{M}_{EL}(e)(m), \ldots\}
\end{array}
$$

Observe that $\mathcal{M}_{EL}$ always produces a monotone function. Given a representation of *AuthMap* in which the authorization of a principal can be found in constant time, it is clear that for any expression e, we can compute $\mathcal{M}_{EL}(e)(m)$ in time linear in the size of the expression. Hence, we can compute $\mathcal{M}_{Assertions}(A)$ and $\mathcal{M}_{Engine}$ in time proportional to the sum of the sizes of the licenses in A.

Using *ExampleLicense*, we can write expressions denoting the first six examples in Figure 15.

```
1)  W
2)  Bob
3)  (lub W Bob)
4)  (glb W Bob)
5)  (glb W Bob Carl)
6)  (glb R (lub (glb Bob Carl)
                (glb Carl Dave)
                (glb Bob Dave)))
```

The last expression shows that this language may be verbose in expressing some monotone functions. Even worse, the last two examples in Figure 15 can not be expressed at all in this language. Of course, by adding more constructs, one could express those examples as well, but at the cost of additional complexity in the trust management engine and possibly additional time taken in computing $\mathcal{M}_{EL}$. The last example would require an existential quantification operator, similar to that of [10, 11].

3 Simple Public Key Infrastructure (SPKI)

Figure 18 shows how to express SPKI [7, 8, 9], including SDSI [12] local names, in the framework of Section 2. The upper part of the figure defines the lattice of SPKI authorizations and the representation of SPKI assertions. The lower part gives the semantics of SPKI assertions, by showing how to map them to assertions in the sense of Section 2. The figure should be read along with Figure 17, which defines the supporting infrastructure of the framework.

The SPKI *Auth* lattice is a powerset lattice with two kinds of authorizations, *name* and *action*. For authmap m, if name authorization $\langle p', n, t\rangle$ is in $m(p)$, then principal p authorizes principal p' to act as name n at time t. Similarly, if action authorization $\langle p', y, t\rangle$ is in $m(p)$ then principal p authorizes principal p' to perform operation y at time t. Elements of *Name* are arbitrary byte strings. Elements of *Sexp* are s-expressions, and denote operations. For this paper, we will leave *Time* unspecified, other than to note that it is totally ordered by $\leq$.

The meaning of a name is always taken relative to a given principal. A *full name* is a principal and a sequence of names, where $\langle p, t\rangle \in \mathcal{M}_{FullP}(f, m)$ means that principal p can act on behalf of full name f at time t. $\mathcal{M}_{FullP}$ is extended to subjects by $\mathcal{M}_{SubjP}$ and to name assertions by $\mathcal{M}_{Name}$. A name assertion is represented by a 4-tuple $\langle p, n, s, \langle t_1, t_2\rangle\rangle$, which means that principal p authorizes subject s to act on behalf of name n at any time between t_1 and t_2. The subject field can either directly identify a full name, or specify a threshold subject of the form $\langle k, \{f_1, f_2, \ldots\}\rangle$. A principal can only act for a threshold subject if at least k of the names $f_1, f_2, \ldots$ denote that principal.

An authorization assertion is represented by a 5-tuple $\langle p, s, d, x, \langle t_1, t_2\rangle\rangle$, which means that principal p authorizes subject s to perform (and delegate, if $d = \mathsf{true}$) the operations allowed by action x at any time between t_1 and t_2. We will not specify actions in more detail – abstractly, they denote sets of s-expressions, so we can assume a function $\mathcal{M}_{Action}$ that gives their meaning. The semantics of authorization assertions is specified by $\mathcal{M}_{Auth}$, which relies on $\mathcal{M}_{SubjA}$ to define the authorizations granted by a subject. The definition of $\mathcal{M}_{Auth}$ formalizes the intuition above by requiring the action x to include the requested operation y and the time period to contain the request time t. If the delegate flag is set, then $\mathcal{M}_{Auth}$ allows requests authorized by the subject; otherwise, it requires the requestor p' to be able to act on behalf of the subject.

Everything is now in place to see how a trust management computation is carried out. Suppose we want to know if principal p authorizes principal p' to perform the operation denoted by y at time t according to the assertions A. We express the request as $u = \{\langle p', y, t\rangle\} \in \mathit{Auth}$. We want to compute $\mathcal{M}_{Engine}(p, u, A)$, which is equivalent to $u \subseteq \mathcal{M}_{Assertions}(A)(p)$. In order to avoid computing the entire least fixpoint, we can specialize the assertions for the given request. Define the restriction of an authorization u' as follows.

$$\mathcal{R}(u') = u' \cap (u \cup \{\langle p'', n, t\rangle \mid p'' \in \mathit{Principal}, n \in \mathit{Name}\})$$

Thus the restriction of an authorization includes no more than current request u and name authorizations at the current time. Extend $\mathcal{R}$ to sets of assertions so that the licenses only produce restricted authorizations as follows.

$$\mathcal{R}(A) = \{\langle p, \lambda m.\, \mathcal{R}(l(m))\rangle \mid \langle p, l\rangle \in A\}$$

We can prove that $u \subseteq \mathcal{M}_{Assertions}(A)(p)$ if and only if $u \subseteq \mathcal{M}_{Assertions}(\mathcal{R}(A))(p)$. We can also observe in Figure 18 that name assertions affect the meaning of authorization assertions, but not vice-versa. The implication of these two facts is that instead of computing a least fixpoint over the *AuthMap* lattice, we can first compute a fixpoint over the lattice $\mathit{Principal} \longrightarrow \mathcal{P}(\mathit{Principal} \times \mathit{Name})$ and then over the lattice $\mathit{Principal} \longrightarrow \mathit{Bool}$. The first fixpoint computes for each principal p the pairs $\langle p', n\rangle$ such that p authorizes p' to act on behalf of name n at time t. The second fixpoint computes for each principal p whether or not the request u is authorized by p. If there are c assertions, the first fixpoint requires $O(c^2)$ space and the second requires only $O(c)$ space.

4 KeyNote

Figure 19 shows how to express KeyNote [2, 3] in the framework of Section 2. The figure should be read along with Figure 17, which defines the supporting infrastructure of the framework. The KeyNote authorizations *Auth* form a function lattice, where an authorization maps a *request* to a *compliance value*. Compliance values are strings, totally ordered, intended to denote levels of authorization. To keep a simple example in mind, imagine that $\mathit{Value} = [\mathsf{false}, \mathsf{true}]$, where $\mathsf{false} \sqsubseteq \mathsf{true}$. The partial order on *Auth* is derived from the order on *Value* viewed as a lattice. A request $\langle P, x\rangle$ consists of a set of requesting principals P (called `ACTION_AUTHORIZERS` in [2]) and an action, which is represented as

a list of pairs of arbitrary strings. The action describes the request; for example, an action to delete a file might be $\{\langle \texttt{operation}, \texttt{delete}\rangle, \langle \texttt{file}, \texttt{/tmp/foo}\rangle\}$.

A KeyNote assertion $\langle p, z, c\rangle$ means that issuing principal p authorizes the requests specified by the conditions c, possibly delegating to licensees z. The licensees language is similar to the toy language of Section 2. It has operators for greatest and least upper bound of the compliance values in the *Value* order, as well as an operation for finding the k-th largest of a set of compliance values. The semantics is summarized by a function $\mathcal{M}_{\textit{Licensees}}$, which takes a licensees expression and a map giving the a compliance value for each principal, and gives the value of the expression. The conditions language has floating point, integer, string, and relational operators for inspecting the request and computing a compliance value. The semantics is summarized by $\mathcal{M}_{\textit{Conditions}}$, which takes a conditions expression and the request, and computes a compliance value. The semantics of KeyNote assertions is given by $\mathcal{M}_{\textit{Keynote}}$, which says that the license corresponding to a KeyNote assertion returns the greatest lower bound (in the *Value* order) of the meaning of the conditions and licensees fields. Delegation is possible via the licensees field, which is only allowed to query the authmap to find other principals' authorizations of the same request $\langle P, x\rangle$. The license field in the assertion is monotone because $\mathcal{M}_{\textit{Licensees}}$ is monotone in its second argument.

A KeyNote trust management engine is responsible for computing a compliance value given a set of assertions and a request. Suppose that the principals P present to principal p the assertions A to prove that the action x is justified, i.e. has a compliance value of at least v. We can express the request as the (partial) function $[\langle P, x\rangle \mapsto v]$. Then, the trust management engine must decide if $\mathcal{M}_{\textit{Engine}}(p, [\langle P, x\rangle \mapsto v], A)$, which is equivalent to $[\langle P, x\rangle \mapsto v] \sqsubseteq \mathcal{M}_{\textit{Assertions}}(A)(p)$, which is equivalent to $v \sqsubseteq \mathcal{M}_{\textit{Assertions}}(A)(p)(P, x)$. Thus, the trust management engine need only compute a fixpoint in the lattice *Principal* $\longrightarrow$ *Value*, recording for each principal its compliance value on the given request.

5 Applications

5.1 Certificate reduction

This section formalizes the notion of certificate reduction, in which a trust management system provides a mechanism to combine several certificates into a single certificate that summarizes their meaning (e.g. SPKI tuple reduction). For example, if we have the assertions

$$\begin{aligned} a_{\mathsf{A1}} &= \langle \mathsf{Alice}, \lambda m.m(\mathsf{Bob})\rangle \\ a_{\mathsf{B}} &= \langle \mathsf{Bob}, \lambda m.m(\mathsf{Carl})\rangle \end{aligned}$$

then we can create the assertion

$$a_{\mathsf{A2}} = \langle \mathsf{Alice}, \lambda m.m(\mathsf{Carl})\rangle$$

and prove that we have not increased the authorizations granted by Alice. Certificate reduction can make the overall system more efficient by reducing the number of certificates carried around and used in subsequent trust engine decisions. It can also provide anonymity to some of the participants. In our example, Bob no longer needs to be mentioned when Alice grants authorizations by delegating to Carl.

We must be careful, however, since it is not the case that for all sets of assertions A that

$$\mathcal{M}_{\textit{Assertions}}(A \cup \{a_{\mathsf{A1}}, a_{\mathsf{B}}\}) \stackrel{?}{=} \mathcal{M}_{\textit{Assertions}}(A \cup \{a_{\mathsf{A2}}\})$$

The following example shows one reason why.

$$\begin{aligned} A &= \{\langle \mathsf{Carl}, \lambda m.\mathsf{W}\rangle\} \\ \mathcal{M}_{\textit{Assertions}}(A \cup \{a_{\mathsf{A1}}, a_{\mathsf{B}}\})(\mathsf{Bob}) &= \mathsf{W} \\ \mathcal{M}_{\textit{Assertions}}(A \cup \{a_{\mathsf{A2}}\})(\mathsf{Bob}) &= \bot \end{aligned}$$

Obviously, because we have removed Bob's delegation to Carl, we have reduced Bob's authorizations. The following example shows why even the authorizations granted by Alice may change.

$$\begin{aligned} A &= \{\langle \mathsf{Bob}, \lambda m.\mathsf{W}\rangle\} \\ \mathcal{M}_{\textit{Assertions}}(A \cup \{a_{\mathsf{A1}}, a_{\mathsf{B}}\})(\mathsf{Alice}) &= \mathsf{W} \\ \mathcal{M}_{\textit{Assertions}}(A \cup \{a_{\mathsf{A2}}\})(\mathsf{Alice}) &= \bot \end{aligned}$$

Since we have removed Alice's delegation to Bob, we have reduced Alice's authorizations.

To simplify the formalization, we will only consider certificate reduction of two certificates. Suppose that we have two assertions $a_1 = \langle p_1, l_1 \rangle$ and $a_2 = \langle p_2, l_2 \rangle$. Define the reduction of a_1 and a_2 as

$$\mathrm{R}(a_1, a_2) = \langle p_1, \lambda m.l_1(m \sqcup [p_2 \mapsto l_2(m)]) \rangle$$

It is possible to prove that the reduced assertion grants no more authorizations than were granted by the original two assertions. That is, for all sets of assertions A,

$$\mathcal{M}_{Assertions}(A \cup \{\mathrm{R}(a_1, a_2)\}) \sqsubseteq \mathcal{M}_{Assertions}(A \cup \{a_1, a_2\})$$

Our earlier examples showed that equality does not hold for arbitrary sets of assertions. However, if none of the assertions mention p_2, then equality does hold for all principals except for p_2. More formally, if for all $\langle p, l \rangle \in A$, $p \neq p_2$ and for all authmaps m, $l(m) = l(m \sqcap [p_2 \mapsto \bot])$, then

$$\begin{aligned} &\mathcal{M}_{Assertions}(A \cup \{\mathrm{R}(a_1, a_2)\}) \\ &\quad = [p_2 \mapsto \bot] \sqcap \mathcal{M}_{Assertions}(A \cup \{a_1, a_2\}) \end{aligned}$$

For a particular trust management system, the language for writing licenses might not be expressive enough to allow certificate reduction. That is, there may be license expressions e_1 and e_2 denoting licenses l_1 and l_2, but no license expression denoting l, where $\langle p_1, l \rangle = \mathrm{R}(\langle p_1, l_1 \rangle, \langle p_2, l_2 \rangle)$. In the case of SPKI, with the minor exception of the intersection of `*range` and `*prefix` actions, the license language is expressive enough. One principle designers of new trust management systems should keep in mind is making the license language powerful enough to express all of the monotone functions that are needed for certificate reduction.

5.2 Proof checking vs. proof construction

One way to view an implementation of $\mathcal{M}_{Engine}$ is that when $\mathcal{M}_{Engine}(p, u, A) = \mathsf{true}$, the engine has constructed a proof that $u \sqsubseteq \mathcal{M}_{Assertions}(A)(p)$. For several reasons, it may make more sense to require the client of the trust engine to provide the proof and have the trust engine simply check that the proof is valid (as is done in [1]). First, the trust engine can be smaller and faster because proof checking is easier than computing the entire fixpoint. Also, third-party modules can be introduced that find the proof using efficient, complex, or new methods. Finally, as part of a trusted computing base, having a simpler and more reliable engine increases overall security.

We can model this approach within our framework using a slightly modified trust engine, $\mathcal{M}_{EngineProof}$, that takes as input a finite sequence of assertions instead of an unordered set of assertions.

$$\mathcal{M}_{EngineProof} : \mathit{Principal} \times \mathit{Auth} \times \mathit{Assertion}^* \longrightarrow \mathit{Bool}$$

To decide if $\mathcal{M}_{EngineProof}(p, u, [a_1, \ldots, a_n])$, where $a_i = \langle p_i, l_i \rangle$, the trust engine computes an increasing sequence of authmaps $m_0, \ldots, m_n$ in the *AuthMap* lattice:

$$\begin{aligned} m_0 &= \bot \\ m_i &= m_{i-1} \sqcup [p_i \mapsto l_i(m_{i-1})] \end{aligned}$$

The engine returns true if $u \sqsubseteq m_n(p)$. It is easy to see that for all i, $m_i \sqsubseteq \mathcal{M}_{Assertions}(A)$, where $A = \{a_1, \ldots, a_n\}$. That is, $\mathcal{M}_{EngineProof}$ computes a lower bound to the fixpoint needed by $\mathcal{M}_{Engine}$. By the pointwise ordering on authmaps, if $m \sqsubseteq \mathcal{M}_{Assertions}(A)$ and $u \sqsubseteq m(p)$ then $u \sqsubseteq \mathcal{M}_{Assertions}(A)(p)$. From this it follows that if $\mathcal{M}_{EngineProof}(p, u, [a_1, \ldots, a_n])$ then $\mathcal{M}_{Engine}(p, u, \{a_1, \ldots, a_n\})$. Thus, $\mathcal{M}_{EngineProof}$ never authorizes a request unless $\mathcal{M}_{Engine}$ would have authorized it.

6 Notation and mathematical background

This section introduces the notation and mathematical background used throughout the paper. Capitalized names in italic font indicate sets, for example, *Int* is the set of integers. Lower case letters typically denote elements of sets, as in $i \in \mathit{Int}$. The power set of the integers is denoted $\mathcal{P}(\mathit{Int})$. Upper case letters typically range over elements of a power set, as in $I \in \mathcal{P}(\mathit{Int})$. The cardinality of a set A is denoted $\mathit{card}(A)$. If A and B are sets, then $A + B$, $A \times B$, and $A \longrightarrow B$ are the disjoint sum, cartesian product, and set of partial functions from A to B, respectively. If A is a set, then A^* denotes the set of finite sequences of elements of A. In expressions denoting sets, the operator precedence (in decreasing order) is

*, $\times$, $\longrightarrow$, $+$. An element of $A \times B \times C$ is denoted $\langle a, b, c\rangle$. An element of A^* is denoted $[a_0, a_1, a_2]$. The expression $[a_1 \mapsto b_1, a_2 \mapsto b_2, \ldots]$ denotes the function f in $A \longrightarrow B$ such that $f(a_1) = b_1$, $f(a_2) = b_2$, ... The expression $\lambda a.\, e$ denotes the function in $A \longrightarrow B$ that when applied to an argument $a \in A$, returns the result of expression e, which may refer to a, and denotes an element of B. For example, $\lambda i.\, i + 1$ is the function that adds one to its argument. The expression $f(a)$ denotes the application of function f to argument a. The inner $\langle\rangle$ is dropped in $f(\langle a, b, c\rangle)$, which is written as $f(a, b, c)$. Function application associates to the left and $\longrightarrow$ associates to the right. For example, if $f = \lambda i.\, \lambda j.\, i - j$, then $f \in \mathit{Int} \longrightarrow \mathit{Int} \longrightarrow \mathit{Int}$ and $f(7)(5) = 2$.

A binary relation $\sqsubseteq$ on A is a subset of $A \times A$. The expression $a \sqsubseteq a'$ denotes $\langle a, a'\rangle \in \sqsubseteq$. The relation $\sqsubseteq$ is reflexive if $a \sqsubseteq a$ for all $a \in A$, transitive if $a \sqsubseteq a''$ whenever $a \sqsubseteq a'$ and $a' \sqsubseteq a''$, and anti-symmetric if $a = a'$ whenever $a \sqsubseteq a'$ and $a' \sqsubseteq a$. A partial order is a set A and a relation $\sqsubseteq$ that is reflexive, transitive, and anti-symmetric. In a partial order, a is an upper bound of a subset $A' \subseteq A$ if for all $a' \in A'$, $a' \sqsubseteq a$. A least upper bound of A', denoted $\bigsqcup A'$, is an upper bound a of A' such that $a \sqsubseteq a_0$ whenever a_0 is an upper bound of A'. The greatest lower bound, denoted $\bigsqcap$, is defined analogously. A lattice is a partial order in which every subset $A' \subseteq A$ has a least upper bound. The least element of a lattice is denoted by $\bot$, and is equal to $\bigsqcup\{\}$. If A is a set then $\mathcal{P}(A)$ is a lattice, where $A' \sqsubseteq A''$ iff $A' \subseteq A''$ and $\bigsqcup\{A_i \mid i \in I\} = \bigcup\{A_i \mid i \in I\}$. If A and B are lattices then $A \times B$ is a lattice, where $\langle a, b\rangle \sqsubseteq \langle a', b'\rangle$ iff $a \sqsubseteq a'$ and $b \sqsubseteq b'$, and $\bigsqcup\{\langle a_i, b_i\rangle \mid i \in I\} = \langle \bigsqcup\{a_i \mid i \in I\}, \bigsqcup\{b_i \mid i \in I\}\rangle$. If A is a set and B is lattice, then $A \longrightarrow B$ is a lattice under the pointwise ordering, where $f \sqsubseteq g$ iff $f(a) \sqsubseteq g(a)$ for all $a \in A$ and where $\bigsqcup\{f_i \mid i \in I\} = \lambda a.\, \bigsqcup\{f_i(a) \mid i \in I\}$.

A function f from a partial order A to a partial order B is *monotone* if $f(a) \sqsubseteq f(a')$ whenever $a \sqsubseteq a'$. The set of monotone functions from A to B is written $A \longrightarrow_m B$. If $f \in A \longrightarrow A$, then a is a fixpoint of f if $f(a) = a$. If A is a partial order, then a *least fixpoint* of f, written $\mathit{lfp}(f)$, is a fixpoint a such that $a \sqsubseteq a'$ whenever a' is a fixpoint of f. If f has a least fixpoint, it is unique. If A is a lattice and $f \in A \longrightarrow_m A$, then f always has a least fixpoint.

A *chain* in a partial order $(A, \sqsubseteq)$ is a sequence of $a_i \in A$ such that for all i, $a_i \sqsubseteq a_{i+1}$. If A and B are partial orders and $f \in A \longrightarrow_m B$, then f is *continuous* if for every chain a_i in A, $f(\bigsqcup\{a_i \mid i \in I\}) = \bigsqcup\{f(a_i) \mid i \in I\}$. Define $f^1(a) = f(a)$ and for all i, $f^{i+1}(a) = f(f^i(a))$. Then, if f is continuous, the least fixpoint of f is given by $\mathit{lfp}(f) = \bigsqcup\{f^i(\bot) \mid i \in \mathit{Int}\}$. If the elements of the lattice A are representable and f is computable and for some i, $f^i(\bot) = f^{i+1}(\bot)$, then this gives us a method for computing $\mathit{lfp}(f)$; namely, compute $f^1(\bot)$, $f^2(\bot)$, $f^3(\bot)$, etc. until the sequence converges.